# Projet1 MLR

Arthur BAUDY
Baptiste THIERRY

Mai 2025

## 1 Introduction

### 1.1 Contexte

La menace inégalée des logiciels malveillants Android est à l'origine de divers problèmes de sécurité sur Internet. L'industrie des malwares Android devient de plus en plus perturbatrice avec près de 12 000 nouvelles instances de malwares Android chaque jour. La détection des logiciels malveillants Android dans les smartphones est une cible essentielle pour que la cyber-communauté se débarrasse des échantillons de logiciels malveillants menaçants.

Les logiciels malveillants Android sont l'une des menaces les plus graves sur Internet, qui a connu une recrudescence sans précédent ces dernières années. C'est un défi ouvert pour les experts en cybersécurité.

La base de données considérée dans ce projet contient une liste plus de 31000 logiciels classés en 12 catégories et 191 familles. L'objectif est de proposer et comparer des algorithmes de machine learning pour identifier automatiquement la catégorie de chacun des logiciels.

Les données sont collectées à partir des fichiers au format Executable Linkage Format et recodées de la façon suivante. Les 2000 premiers octets du fichier sont extraits. Des 0 sont ajoutés si le fichier n'est pas assez long. Ces chaînes ASCII sont ensuite encodées par un simple code de chiffrement pour supprimer les informations sensibles et transmises à un encodeur base64 pour obtenir des représentations radix-64 lisibles.

La catégorie et la famille de chaque logiciel est déterminée par des moteurs antivirus.

Chaque ligne du fichier de données correspond donc à un programme exécutable. Les 2493 premières colonnes sont le résultat du chiffrage. La colonne 2494 est le hash (identifiant). Et les 3 dernières colonnes donnent des informations sur le type de logiciel (famille, malveillant ou non).

### 1.2 Plan

- Présentation des données et standardisation
- ACP
- Méthodes K-PPV et SVM
- Arbres et forêts
- Réseaux de neurones
- Comparaison

## 2 Présentation des données

### 2.1 Choix de la classification

Notre étude ne portera pas sur l'ensemble total des données, puisque celui-ci est trop important. On fait le choix de p=0.25, pour obtenir comme ensemble de données : Nombre d'individus : 2978, nombre de variables : 2497.

On commence par afficher la répartition des malwares, catégories et familles.

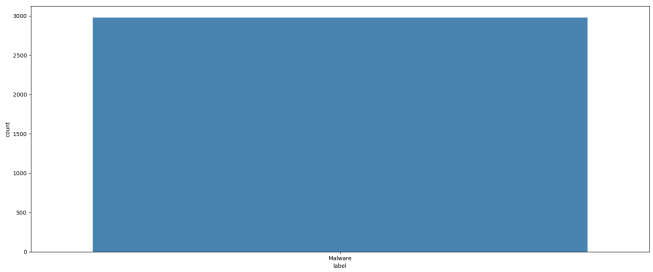

Figure 1: Répartitions des malwares

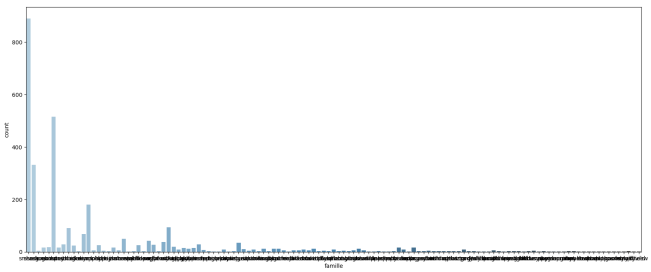

Figure 2: Répartition des familles

On observe que tous les individus possèdent le label malware. Les familles sont quant à elles trop nombreuses et une grand majorité ont une répartition très faible. Pour les catégories, elles sont au nombre de 12 et leur répartition est inégale.

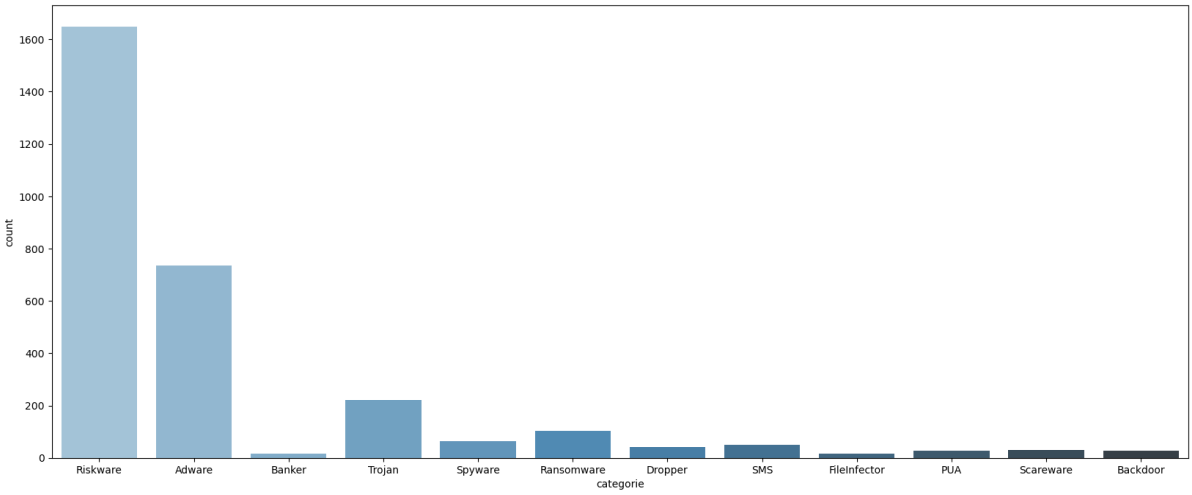


Figure 3: Répartition des catégories

Comme la prédiction de la variable malware ne présente pas d'intérêt ici et que la variable famille semble difficile à illustrer, nous décidons de porter notre étude sur la classification de la variable catégories.

## 2.2 Vérification et préparation des données

On supprime de nos données les variables Malwares, Familles et Hash, puis on vérifie qu'il n'y a pas de données manquantes.

La variable à prédire y est la variable représentant les catégories parmis [Riskware, Adware, Bankern, Trojan, Spyware, Ransomware, Dropper, SMS, FileInfector, PUA, Scareware, Backdoor] et on obtient un X de dimension n=2978 et p=2493.

Pour pouvoir évaluer nos méthodes de prédiction, on sépare notre jeu de données en 70 % de train et 30% de test avec le paramètre stratify=y pour que toutes les catégories soient représentées. On vérifie que la répartition dans le train et dans le test est semblable, pour ne pas se retrouver avec des catégories avec une proportion trop faible, voir sans individu.

Pourcentages de chaque catégorie dans l'ensemble complet, d'apprentissage et de test. 24.71%,24.71% ,24.72% : Adware 0.94%,0.96% ,0.89% : Backdoor 0.54%,0.53% ,0.56% : Banker 1.38%,1.39% ,1.34% : Dropper 0.5%,0.53% ,0.45% : FileInfector 0.91%,0.91% ,0.89% : PUA 3.46%,3.45% ,3.47% : Ransomware 55.34%,55.33% ,55.37% : Riskware 1.65%,1.63% ,1.68% : SMS 1.01%,1.01% ,1.01% : Scareware 2.12%,2.11% ,2.13% : Spyware 7.45%,7.44% ,7.49% : Trojan.

Enfin, on vérifie l'amplitude des variables. Les variables 0, 3, 4, 5, 6, 7, 8, 10 et 11 ne représentent pas des valeurs de bits contrairement aux autres. L'amplitude peut être élevée pour certaines, on décide donc de standardiser les données.

## 2.3 ACP

Après avoir standardisé nos données, on décide d'appliquer l'ACP dans l'objectif de réduire nos dimensions et d'améliorer nos temps de calcul sur les différentes méthodes.

Après avoir construit les graphiques des variances cumulées, on remarque qu'après les cent premières composantes, elles ne représentent que très peu de variance expliquée. On fait le choix de conserver 95% de variance expliquée, ce qui nous donne un nombre de 129 variables à conserver.

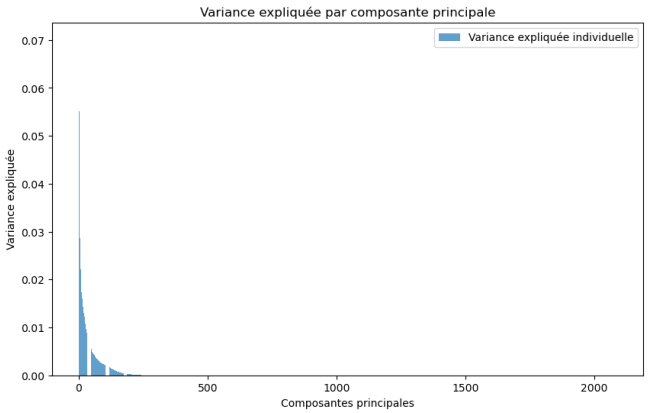


Figure 4: Graphique des variances expliquées

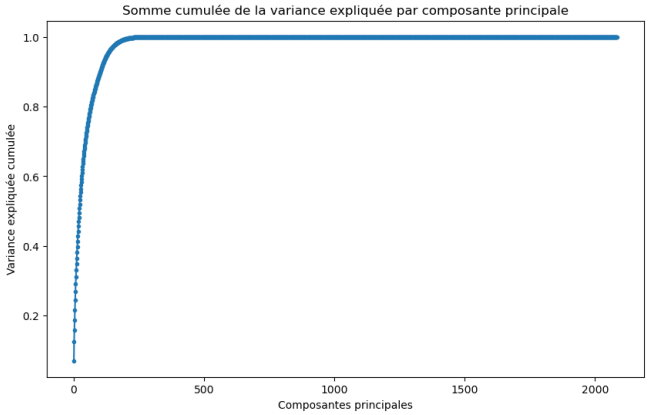


Figure 5: Variances expliquées cumulatives

On termine par représenter nos individus dans les plans obtenus par les composantes de l'ACP. Pour rendre les nuages de points plus visibles, on ajuste l'échelle quitte à ne pas rendre visible quelques rares points très éloignés du centre.

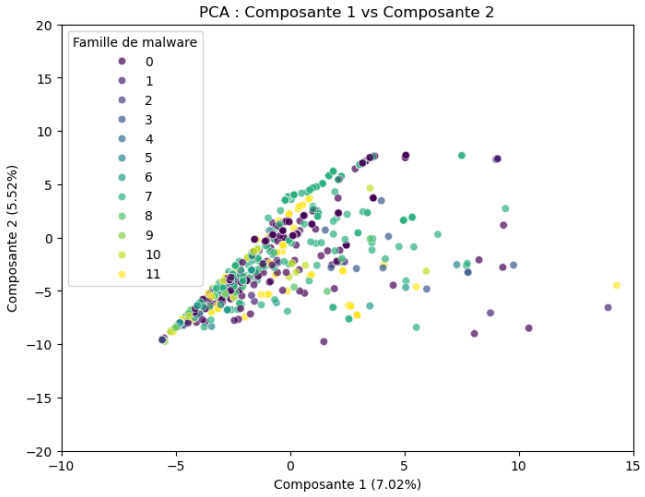


Figure 6: Projection dans le plan (PC1,PC2)

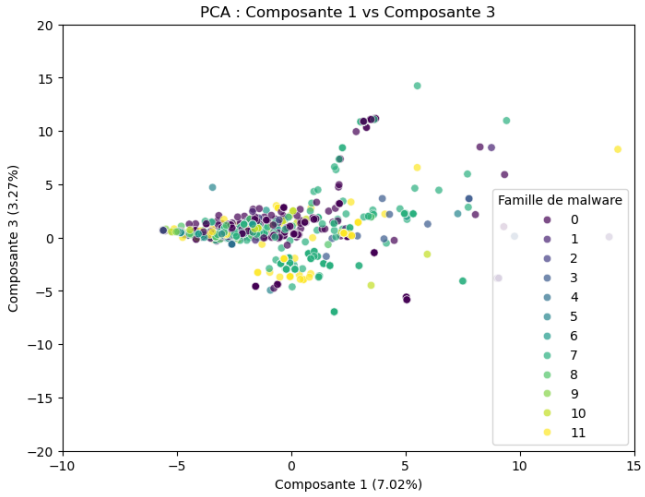


Figure 7: Projection dans le plan (PC1,PC3)

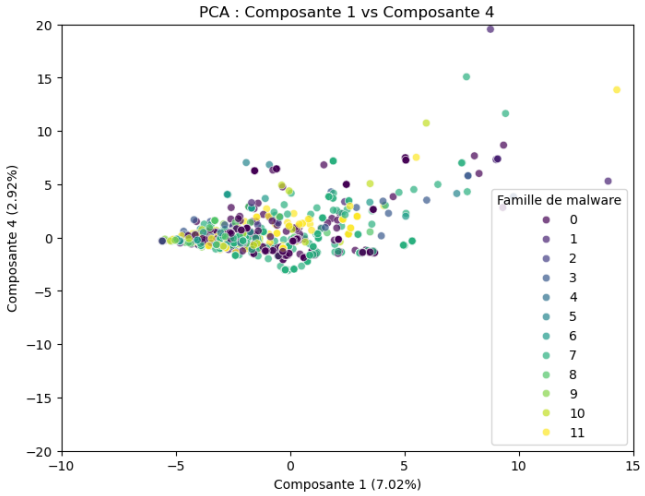


Figure 8: Projection dans le plan (PC1,PC4)

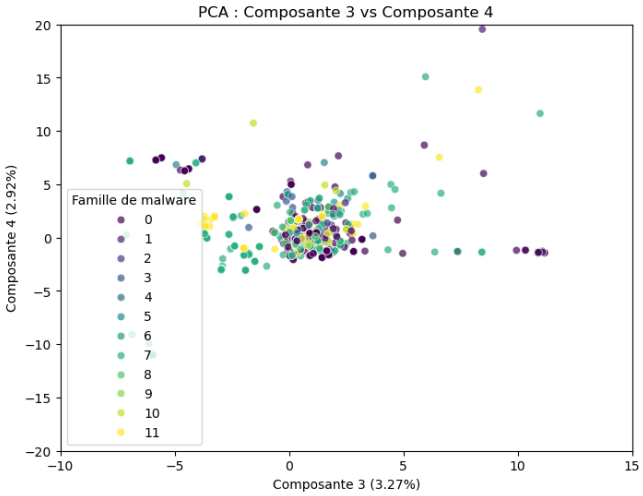


Figure 9: Projection dans le plan (PC3,PC4)

# 3 k-ppv et SVM

## 3.1 k-ppv

Dans cette partie, nous nous intéressons à l'algorithme des k plus proches voisins. On trouve les meilleurs paramètres (le nombre de voisins, le poids et la métrique) par cross-validation avec un nombre de pli égale à 10 et une taille de l'ensemble de test de 30%. On trouve alors comme paramètre :

- Métrique : Euclidienne
- Nombre de voisins : 10
- Poids : 'distance'

Pour observer performances, on regarde la matrice de confusion

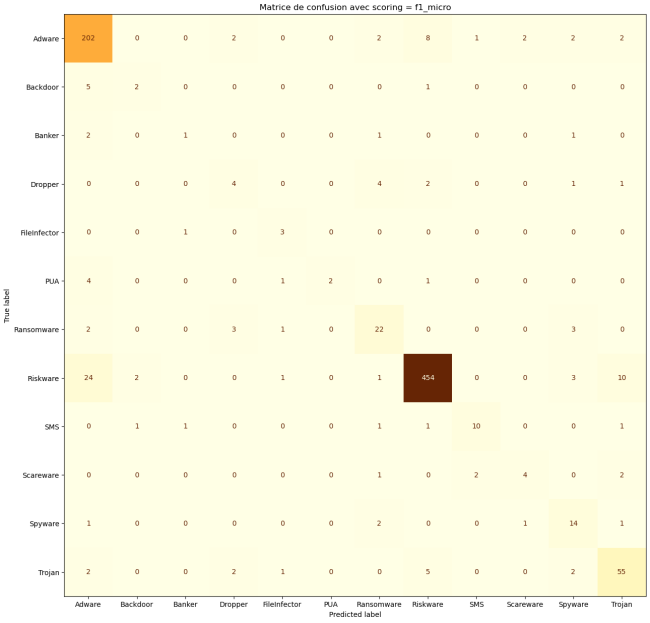


Figure 10: Matrice de confusion K-PPV

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Adware | 0.83 | 0.91 | 0.87 | 221 |
| Backdoor | 0.40 | 0.25 | 0.31 | 8 |
| Banker | 0.33 | 0.20 | 0.25 | 5 |
| Dropper | 0.36 | 0.33 | 0.35 | 12 |
| FileInfector | 0.43 | 0.75 | 0.55 | 4 |
| PUA | 1.00 | 0.25 | 0.40 | 8 |
| Ransomware | 0.65 | 0.71 | 0.68 | 31 |
| Riskware | 0.96 | 0.92 | 0.94 | 495 |
| SMS | 0.77 | 0.67 | 0.71 | 15 |
| Scareware | 0.57 | 0.44 | 0.50 | 9 |
| Spyware | 0.54 | 0.74 | 0.62 | 19 |
| Trojan | 0.76 | 0.82 | 0.79 | 67 |
| accuracy | | | 0.86 | 894 |
| macro avg | 0.63 | 0.58 | 0.58 | 894 |
| weighted avg | 0.87 | 0.86 | 0.86 | 894 |

Figure 11: Scores de prédiction K-PPV

On voit que les classes avec beaucoup de représenter ont des bons scores mais ça n'est pas le cas des catégories peu peuplées. En effet, par exemple la catégorie Trojan avec 67 représentants a une précision de 0.76 ce qui suggère peu de faux positifs et un recall de 0.82 ce qui signifie qu'il y a peu de faux négatifs mais ça n'est pas le cas de la catégorie Backdoor qui n'a seulement 8 individus. Cela est dû notamment à l'algorithme des k-ppv car si une catégorie est peu représenté alors il y des chances que le nombre de voisins d'un individus appartiennent à une autre classe que celle de l'individu en question.

### 3.2 SVM

Dans cette partie, on applique la méthode des machines à vecteurs de support. On détermine les meilleurs paramètres par cross-validation avec un nombre de pli égale à 5. On trouve les paramètres :

- Kernel : rbf
- C : 1000

On ajoute en plus le paramètre class_weight='balanced' pour contrebalancer l'écart de proportions entre les classes et utiliser ce paramètre permettait d'obtenir des résultats meilleurs.

Comme précédemment, on affiche la matrice de confusion et les scores obtenus.

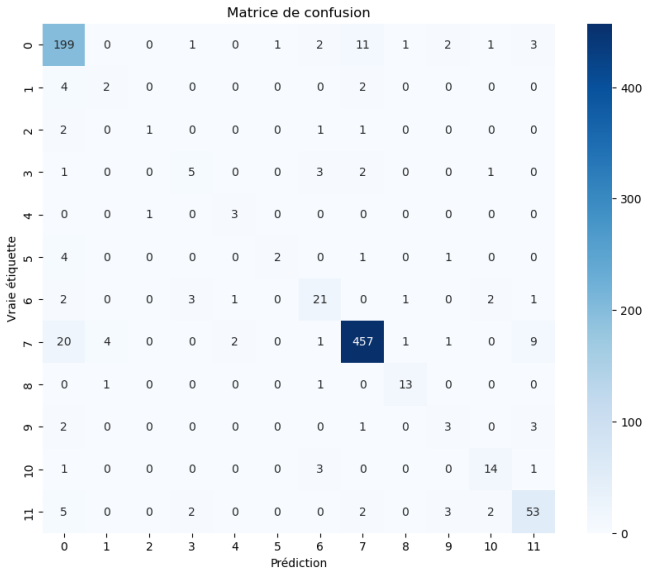


Figure 12: Matrice de confusion SVM

Rapport de classification :

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.83 | 0.90 | 0.86 | 221 |
| 1 | 0.29 | 0.25 | 0.27 | 8 |
| 2 | 0.50 | 0.20 | 0.29 | 5 |
| 3 | 0.45 | 0.42 | 0.43 | 12 |
| 4 | 0.50 | 0.75 | 0.60 | 4 |
| 5 | 0.67 | 0.25 | 0.36 | 8 |
| 6 | 0.66 | 0.68 | 0.67 | 31 |
| 7 | 0.96 | 0.92 | 0.94 | 495 |
| 8 | 0.81 | 0.87 | 0.84 | 15 |
| 9 | 0.30 | 0.33 | 0.32 | 9 |
| 10 | 0.70 | 0.74 | 0.72 | 19 |
| 11 | 0.76 | 0.79 | 0.77 | 67 |
| accuracy | | | 0.86 | 894 |
| macro avg | 0.62 | 0.59 | 0.59 | 894 |
| weighted avg | 0.87 | 0.86 | 0.86 | 894 |

Précision : 0.8646532438478747

Figure 13: Scores de prédiction SVM

On remarque que les scores sont très similaires à ceux obtenus pour k-ppv. Les scores de chaque classe peuvent varier, mais les moyennes des scores sont identiques.

## 4 Arbres et Forêts

Dans cette section, on a considéré l'ensemble du jeux de données car les temps de calculs le permettent et les résultats sont plus intéressant. On a donc 2493 variables pour 12026 individus. Le but de cette partie est de construire un prédicteur de plus en plus performants en partant d'un simple arbre jusqu'à une agrégation de deux forêts.

### 4.1 Arbres

Dans cette partie, on s'intéresse à la construction d'arbres de décisions sur notre jeux de données. On commence par chercher les meilleurs paramètres par cross validation, et on trouve

- Profondeur maximum : 30

- Nombre minimum d'échantillons par feuille : 2
- Séparation : 'best'

On va chercher maintenant à élaguer l'arbre obtenu pour éviter le sur-apprentissage. On compare les différents arbres élagué suivant leur f1 score.

Et on trouve alors que la meilleur valeur de alpha est 0.0001466594242259642

On peut alors comparer le nombre de noeuds et la profondeur de nos arbres.

| | Arbre de base | Arbre élagué |
|---|---|---|
| Nombre de noeux | 851 | 633 |
| Profondeur | 21 | 16 |

Puis pour pouvoir observer nos résultats, on affiche la matrice de confusion et les scores

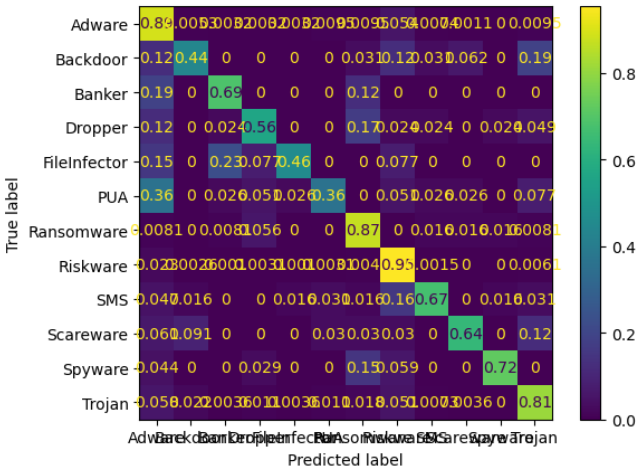


Figure 14: Matrice de confusion arbre de prédiction

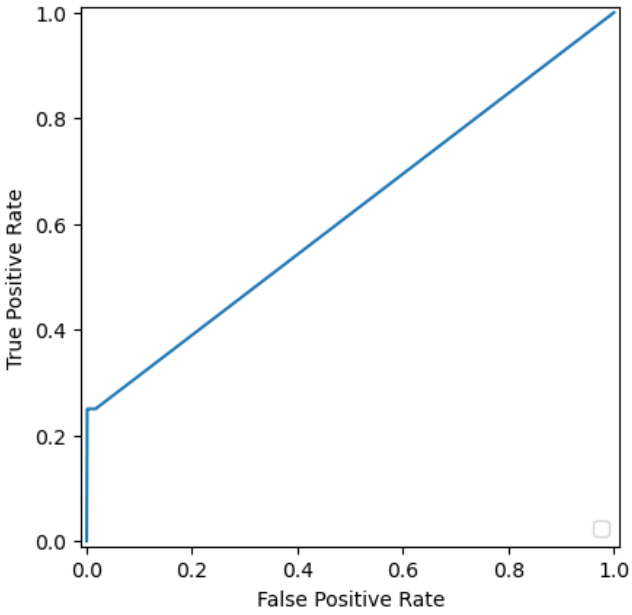


Figure 15: Courbe ROC arbre de prédiction

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Adware | 0.90 | 0.89 | 0.90 | 950 |
| Backdoor | 0.41 | 0.44 | 0.42 | 32 |
| Banker | 0.48 | 0.69 | 0.56 | 16 |
| Dropper | 0.49 | 0.56 | 0.52 | 41 |
| FileInfector | 0.43 | 0.46 | 0.44 | 13 |
| PUA | 0.40 | 0.36 | 0.38 | 39 |
| Ransomware | 0.71 | 0.87 | 0.78 | 124 |
| Riskware | 0.95 | 0.95 | 0.95 | 1953 |
| SMS | 0.72 | 0.67 | 0.69 | 64 |
| Scareware | 0.75 | 0.64 | 0.69 | 33 |
| Spyware | 0.92 | 0.72 | 0.81 | 68 |
| Trojan | 0.85 | 0.81 | 0.83 | 275 |
| accuracy | | | 0.89 | 3608 |
| macro avg | 0.67 | 0.67 | 0.67 | 3608 |
| weighted avg | 0.90 | 0.89 | 0.89 | 3608 |

Figure 16: Scores de l'arbre de prédiction

Encore une fois, on voit que les scores sur les catégorie sous-représentées sont plutôt mauvais. Dans ce qui suit on va essayer de régler ce problème en utilisant plutôt des forêts mais on commence par essayer de faire deux arbres avec un reconnaissant les catégorie très représentées puis un reconnaissant les petites catégories.

## 4.2 Deux arbres de prédictions

Comme dit précédemment, on va utiliser deux arbres pour essayer d'améliorer notre prédicteur. Pour cela, on entraîne un premier arbres sur nos données de train en regroupant les catégories Backdoor,Banker,Dropper,FileInfector,PUA,Ranson dans une seule catégorie commune et on entraine un second arbre sur nos données de train en ne gardant que les catégorie citées précédemment.

Pour le premier arbres, on a trouvé les paramètres suivant par cross-validation

- Profondeur maximum : 35
- Nombre minimum d'échantillons par feuille : 2
- Séparation : 'best'

Puis on applique l'élagage à cet arbre

| | Arbre de base | Arbre élagué |
|---|---|---|
| Nombre de noeux | 561 | 417 |
| Profondeur | 21 | 13 |

Et on affiche les résultats


Figure 17: Matrice de confusion de l'arbre sur les grandes catégories

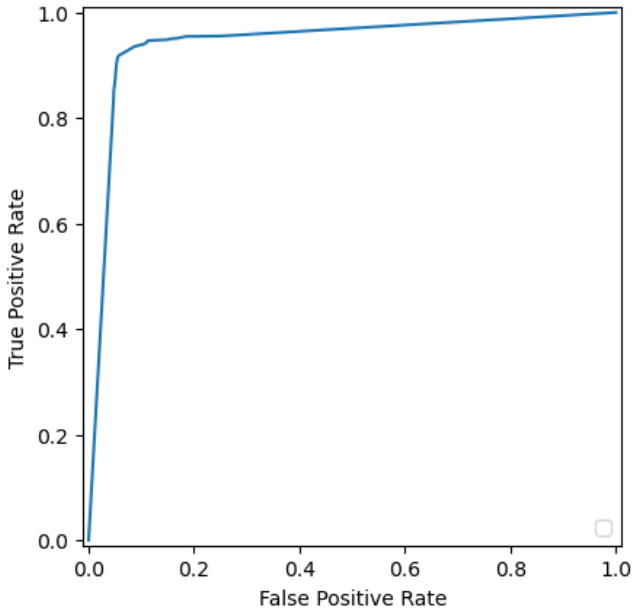


Figure 18: Courbe ROC de l'arbre sur les grandes catégories

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Adware | 0.87 | 0.87 | 0.87 | 665 |
| Riskware | 0.94 | 0.93 | 0.93 | 1367 |
| Trojan | 0.76 | 0.75 | 0.76 | 192 |
| commun_class | 0.77 | 0.80 | 0.79 | 301 |
| accuracy | | | 0.88 | 2525 |
| macro avg | 0.83 | 0.84 | 0.84 | 2525 |
| weighted avg | 0.89 | 0.88 | 0.88 | 2525 |

Figure 19: Score de l'arbre sur les grandes catégories

On fait exactement la même chose pour le deuxième arbres avec comme paramètres

- Profondeur maximum : 20
- Nombre minimum d'échantillons par feuille : 2
- Séparation : 'best'

et on obtient

| | Arbre de base | Arbre élagué |
|---|---|---|
| Nombre de noeux | 173 | 157 |
| Profondeur | 12 | 12 |

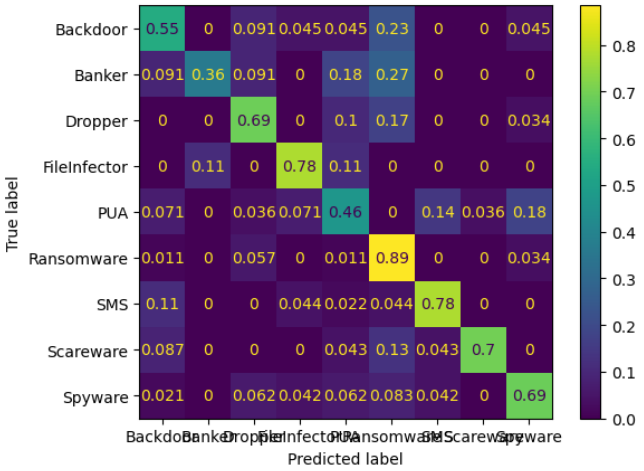


Figure 20: Matrice de confusion de l'arbre sur les petites catégorie

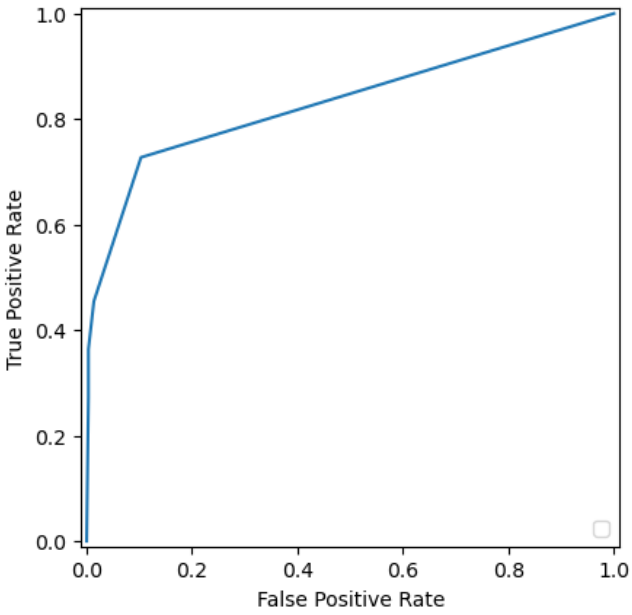


Figure 21: Courbe ROC de l'arbre sur les petites catégories

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Backdoor | 0.50 | 0.55 | 0.52 | 22 |
| Banker | 0.80 | 0.36 | 0.50 | 11 |
| Dropper | 0.62 | 0.69 | 0.66 | 29 |
| FileInfector | 0.50 | 0.78 | 0.61 | 9 |
| PUA | 0.50 | 0.46 | 0.48 | 28 |
| Ransomware | 0.78 | 0.89 | 0.83 | 87 |
| SMS | 0.83 | 0.78 | 0.80 | 45 |
| Scareware | 0.94 | 0.70 | 0.80 | 23 |
| Spyware | 0.77 | 0.69 | 0.73 | 48 |
| accuracy | | | 0.72 | 302 |
| macro avg | 0.69 | 0.65 | 0.66 | 302 |
| weighted avg | 0.73 | 0.72 | 0.72 | 302 |

Figure 22: Score de l'arbre sur les petites catégories

On remarque que ces deux arbres sont meilleurs sur leurs jeux de données que celui fait dans la section précédentes car ils ont moins de catégories à prédire. Enfin, on applique le premier arbre sur les données de test et on applique le deuxième arbre sur les données ayant été classifiées dans la catégorie commune. On obtient alors

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Adware | 0.88 | 0.89 | 0.88 | 950 |
| Backdoor | 0.26 | 0.25 | 0.25 | 32 |
| Banker | 0.71 | 0.62 | 0.67 | 16 |
| Dropper | 0.51 | 0.54 | 0.52 | 41 |
| FileInfector | 0.47 | 0.54 | 0.50 | 13 |
| PUA | 0.35 | 0.33 | 0.34 | 39 |
| Ransomware | 0.72 | 0.85 | 0.78 | 124 |
| Riskware | 0.95 | 0.94 | 0.95 | 1953 |
| SMS | 0.75 | 0.72 | 0.74 | 64 |
| Scareware | 0.59 | 0.70 | 0.64 | 33 |
| Spyware | 0.88 | 0.78 | 0.83 | 68 |
| Trojan | 0.79 | 0.77 | 0.78 | 275 |
| accuracy | | | 0.88 | 3608 |
| macro avg | 0.66 | 0.66 | 0.66 | 3608 |
| weighted avg | 0.88 | 0.88 | 0.88 | 3608 |

Figure 23: Score de l'agrégation des deux arbres

On remarque une petite amélioration des scores grâce à cette agrégation de ces deux arbres. Dans la suite, on va continuer avec l'agrégation de modèle pour encore améliorer notre prédicteur.

## 4.3 Forêts

Au vu des résultats obtenus dans la partie précédentes, l'agrégation de modèle devrait pouvoir encore nous faire gagner en score. On commence par une forêt puis on fait comme dans la partie précédentes avec deux forêts pour prédire les grandes catégorie puis les petites catégories. Nous avons tenté d'appliquer les méthodes AdaBoost et GradientBoosting mais sur un jeu de données trop petit ça n'était pas concluant et sur un plus grand jeu de données, les temps de calculs étaient beaucoup trop long.

### 4.3.1 Forêt simple

On commence par chercher les paramètres optimaux par cross-validation sur l'ensemble de données d'entraînement et on obtient

- Profondeur maximum : 35
- Nombre minimum d'échantillons par feuille : 2
- Séparation : 'best'

Puis on affiche les résultats sur les données d'entraînement

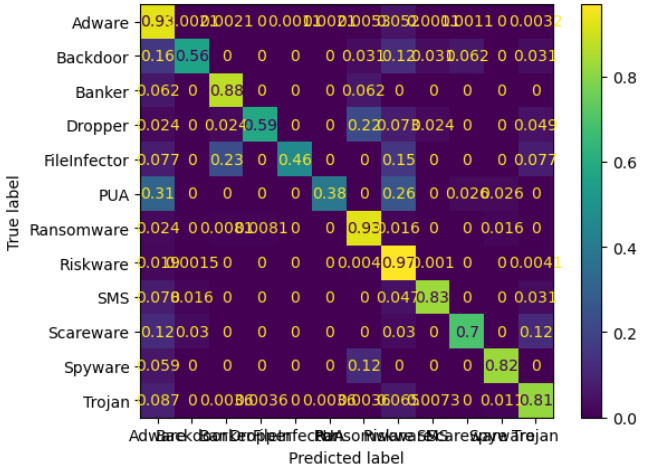


Figure 24: Matrice de confusion de la forêt

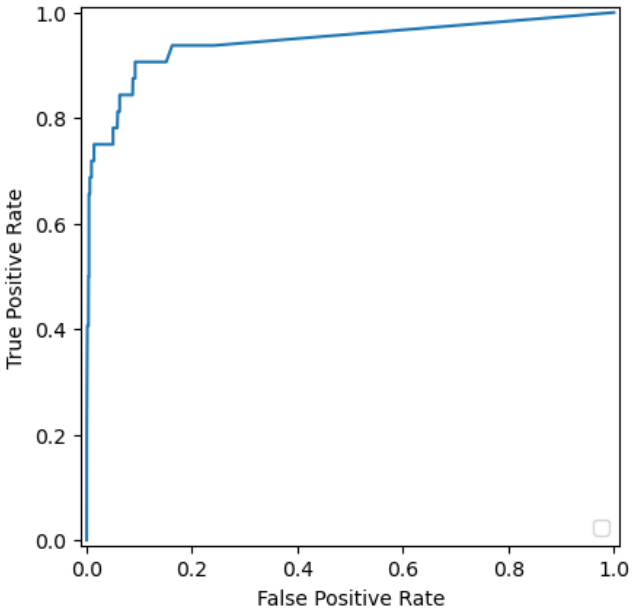


Figure 25: Courbe ROC de la forêt

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Adware | 0.90 | 0.93 | 0.92 | 950 |
| Backdoor | 0.72 | 0.56 | 0.63 | 32 |
| Banker | 0.64 | 0.88 | 0.74 | 16 |
| Dropper | 0.92 | 0.59 | 0.72 | 41 |
| FileInfector | 0.86 | 0.46 | 0.60 | 13 |
| PUA | 0.83 | 0.38 | 0.53 | 39 |
| Ransomware | 0.78 | 0.93 | 0.85 | 124 |
| Riskware | 0.95 | 0.97 | 0.96 | 1953 |
| SMS | 0.88 | 0.83 | 0.85 | 64 |
| Scareware | 0.85 | 0.70 | 0.77 | 33 |
| Spyware | 0.90 | 0.82 | 0.86 | 68 |
| Trojan | 0.91 | 0.81 | 0.86 | 275 |
| accuracy | | | 0.92 | 3608 |
| macro avg | 0.85 | 0.74 | 0.77 | 3608 |
| weighted avg | 0.92 | 0.92 | 0.92 | 3608 |

Figure 26: Score de la forêt

On remarque qu'il y a encore beaucoup d'erreurs sur les petites classes mais il y a une nette amélioration par rapport aux parties précédentes. Le score de précision des différentes variables est meilleur en moyenne ce qui suggère moins de faux positifs et le score recall est aussi meilleur suggérant moins de faux négatifs. On notera aussi que la catégorie Adware est "absorbante" car beaucoup d'individus sont prédit comme étant dans cette classe alors que ça n'est pas le cas.

### 4.3.2 Agrégation de deux forêts

Comme pour la partie avec les deux arbres, on regroupe les catégories Backdoor, Banker, Dropper, FileInfector, PUA, Ransomware, SMS, Scareware, Spyware dans une seule catégorie commune puis on entraîne une première forêts sur ces données puis on entraîne une deuxième forêt sur les petites classes.

Pour la première forêts, avec 35 features max et 2 échantillons minimums par feuilles trouvés par cross-validation, sur les grandes classes, on obtient

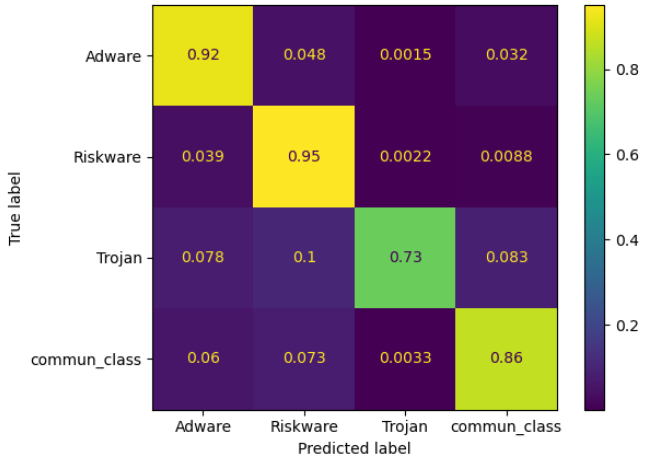


Figure 27: Matrice de confusion de la forêt sur les grandes classes

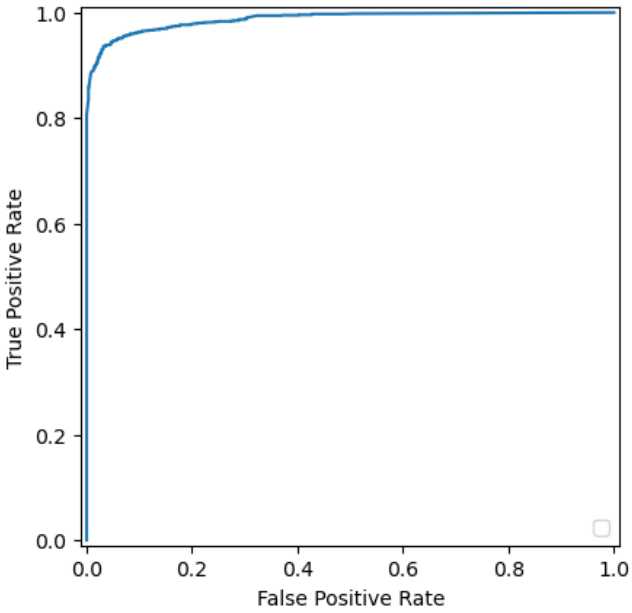


Figure 28: Courbe ROC de la forêt sur les grandes classes

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Adware | 0.88 | 0.92 | 0.90 | 665 |
| Riskware | 0.95 | 0.95 | 0.95 | 1367 |
| Trojan | 0.97 | 0.73 | 0.83 | 192 |
| commun_class | 0.84 | 0.86 | 0.85 | 301 |
| accuracy | | | 0.92 | 2525 |
| macro avg | 0.91 | 0.87 | 0.88 | 2525 |
| weighted avg | 0.92 | 0.92 | 0.91 | 2525 |

Figure 29: Score de la forêt sur les grandes classe

Pour la deuxième forêts, avec 30 features max et 2 échantillons minimums par feuilles trouvés par cross-validation, sur les petites classes

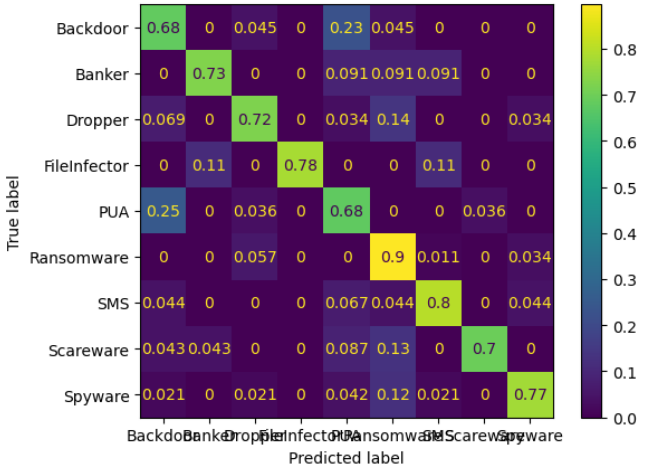


Figure 30: Matrice de confusion de la forêt

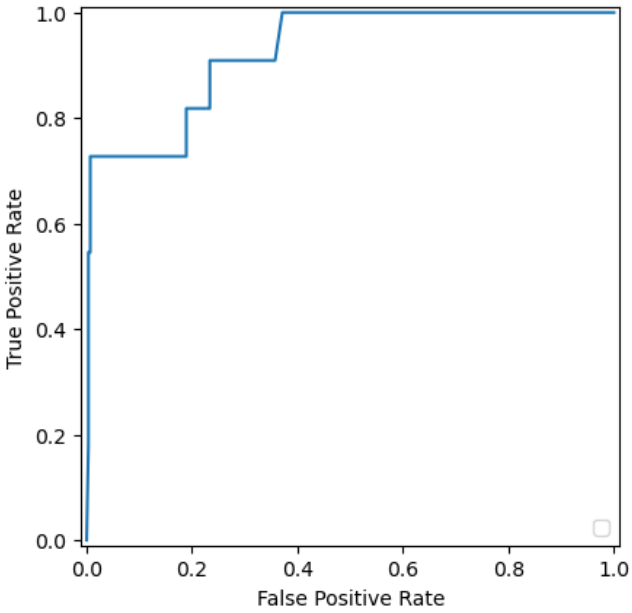


Figure 31: Courbe ROC de la forêt

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Backdoor | 0.54 | 0.68 | 0.60 | 22 |
| Banker | 0.80 | 0.73 | 0.76 | 11 |
| Dropper | 0.72 | 0.72 | 0.72 | 29 |
| FileInfector | 1.00 | 0.78 | 0.88 | 9 |
| PUA | 0.58 | 0.68 | 0.62 | 28 |
| Ransomware | 0.82 | 0.90 | 0.86 | 87 |
| SMS | 0.90 | 0.80 | 0.85 | 45 |
| Scareware | 0.94 | 0.70 | 0.80 | 23 |
| Spyware | 0.86 | 0.77 | 0.81 | 48 |
| accuracy | | | 0.78 | 302 |
| macro avg | 0.80 | 0.75 | 0.77 | 302 |
| weighted avg | 0.80 | 0.78 | 0.79 | 302 |

Figure 32: Score de la forêt

Pour un score final de l'agrégation des deux forêts sur l'ensemble test

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Adware | 0.91 | 0.92 | 0.91 | 950 |
| Backdoor | 0.51 | 0.56 | 0.54 | 32 |
| Banker | 0.64 | 0.88 | 0.74 | 16 |
| Dropper | 0.71 | 0.61 | 0.66 | 41 |
| FileInfector | 0.70 | 0.54 | 0.61 | 13 |
| PUA | 0.50 | 0.36 | 0.42 | 39 |
| Ransomware | 0.74 | 0.92 | 0.82 | 124 |
| Riskware | 0.96 | 0.96 | 0.96 | 1953 |
| SMS | 0.76 | 0.91 | 0.83 | 64 |
| Scareware | 0.64 | 0.76 | 0.69 | 33 |
| Spyware | 0.82 | 0.79 | 0.81 | 68 |
| Trojan | 0.95 | 0.76 | 0.84 | 275 |
| accuracy | | | 0.91 | 3608 |
| macro avg | 0.74 | 0.75 | 0.74 | 3608 |
| weighted avg | 0.92 | 0.91 | 0.91 | 3608 |

Figure 33: Score de l'agrégation des deux forêts

On voit que les scores ne se sont pas vraiment améliorés par rapport à la forêt précédente, on voudra plutôt alors considérer la forêt faite à la section précédente.

# 5 Réseau de neurones

## 5.1 Choix des paramètres du modèle

A partir de l'ensemble test, on conserve 20 % des données pour obtenir l'ensemble de validation du réseau de neurones, les 80 % restant (noté train2) serviront pour l'entraînement du modèle. Avant de déterminer notre meilleur modèle, nous avons du essayer plusieurs configurations de couches et de paramètres associés :

- En couche d'entrée nous avons essayé les couches Conv1D ou Dense. Dans le cas de Conv1D, on conservait les paramètres kernel_size=3, activation='relu', input_shape=(X_train2.shape[1], 1)). Le nombre de filtres pouvait varier, comme le nombre de neurones dans le choix d'une couche Dense.
- En couche de sortie, nous avons conservé cette couche Dense(y_train2.shape[1], activation='softmax')) pour avoir un nombre de neurones égal à nos 12 classes et le parmètre softmax est adapté pour de la classification.
- Entre les deux, nous avons essayé les couches suivantes : Conv1D, MaxPooling1D, Dense, Dropout et Flaten. Soit en les ajoutant, les retirant ou en changeant l'ordre. On ajoutait ou non à la suite d'une couche Conv1D, une couche MaxPooling1D avec pool_size=2. Pour Dense, on a conservé le paramètre activation='relu'. On faisait varier le nombre de neurones(64, 128, 256, ...) et on activait ou non un paramètre de régularisation l2. Les valeurs testées pour la couche Dropout étaient 0.2, 0.25, 0.3 ou 0.5.
- Pour la compilation, on a choisi : compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
- Et pour l'entraînement : fit(X_train2, y_train2, epochs=40, # (parfois moins lors des essais) batch_size=16, # (aussi 8 et 32) validation_data=(X_val, y_val), class_weight=class_weights_dict, # (Pour combler l'écart des proportions des classe, mais les résultats étaient souvent inférieurs lorsqu'activé) callbacks=[checkpoint, early_stopping ])

Comme la cross-validation est coûteuse pour un réseau de neurones et que nous n'avons pas réussi à la mettre en place, nous avons choisi d'utiliser la fonction ModelCheckpoint avec ces paramètres : ModelCheckpoint( filepath='bestmodel.keras', monitor='valaccuracy', savebestonly=True, verbose=1 ) ce qui permet de sauvegarder les configurations de notre modèle lorsque celles-ci étaient plus performantes selon la valeur de précision sur les données de validation. Nous avons aussi utilisé la fonction EarlyStopping pour arrêter l'entraînement prématurément lorque les résultats de perte sur l'ensemble de validation ne s'améliorient pas après plusieurs époques.

## 5.2 Meilleur modèle

Voici les couches utilisées par le meilleur modèle, ainsi que leurs configurations en détail.

Model: "sequential_5"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv1d_6 (Conv1D) | (None, 147, 32) | 128 |
| max_pooling1d (MaxPooling1D) | (None, 73, 32) | 0 |
| dense_10 (Dense) | (None, 73, 256) | 8,448 |
| dropout_5 (Dropout) | (None, 73, 256) | 0 |
| flatten_5 (Flatten) | (None, 18688) | 0 |
| dense_11 (Dense) | (None, 12) | 224,268 |

Total params: 698,534 (2.66 MB)
Trainable params: 232,844 (909.55 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 465,690 (1.78 MB)

Figure 34: Couches du réseau de neurones


```
Configurations des couches :
Couche 0: conv1d_6
Configuration: {'name': 'conv1d_6', 'trainable': True, 'dtype': {'module': 'keras', 'class_name': 'DTypePolicy',
'config': {'name': 'float32'}, 'registered_name': None}, 'filters': 32, 'kernel_size': (3,), 'strides': (1,),
'padding': 'valid', 'data_format': 'channels_last', 'dilation_rate': (1,), 'groups': 1, 'activation': 'relu',
'use_bias': True, 'kernel_initializer': {'module': 'keras.initializers', 'class_name': 'GlorotUniform',
'config': {'seed': None}, 'registered_name': None}, 'bias_initializer': {'module': 'keras.initializers',
'class_name': 'Zeros', 'config': {}, 'registered_name': None}, 'kernel_regularizer': None, 'bias_regularizer':
None, 'activity_regularizer': None, 'kernel_constraint': None, 'bias_constraint': None}
---
Couche 1: max_pooling1d
Configuration: {'name': 'max_pooling1d', 'trainable': True, 'dtype': {'module': 'keras', 'class_name':
'DTypePolicy', 'config': {'name': 'float32'}, 'registered_name': None}, 'pool_size': (2,), 'padding': 'valid',
'strides': (2,), 'data_format': 'channels_last'}
---
Couche 2: dense_10
Configuration: {'name': 'dense_10', 'trainable': True, 'dtype': {'module': 'keras', 'class_name': 'DTypePolicy',
'config': {'name': 'float32'}, 'registered_name': None}, 'units': 256, 'activation': 'relu', 'use_bias': True,
'kernel_initializer': {'module': 'keras.initializers', 'class_name': 'GlorotUniform', 'config': {'seed': None},
'registered_name': None}, 'bias_initializer': {'module': 'keras.initializers', 'class_name': 'Zeros', 'config':
{}, 'registered_name': None}, 'kernel_regularizer': {'module': 'keras.regularizers', 'class_name': 'L2',
'config': {'l2': 0.01}, 'registered_name': None}, 'bias_regularizer': None, 'kernel_constraint': None,
'bias_constraint': None}
---
Couche 3: dropout_5
Configuration: {'name': 'dropout_5', 'trainable': True, 'dtype': {'module': 'keras', 'class_name':
'DTypePolicy', 'config': {'name': 'float32'}, 'registered_name': None}, 'rate': 0.25, 'seed': None,
'noise_shape': None}
---
Couche 4: flatten_5
Configuration: {'name': 'flatten_5', 'trainable': True, 'dtype': {'module': 'keras', 'class_name':
'DTypePolicy', 'config': {'name': 'float32'}, 'registered_name': None}, 'data_format': 'channels_last'}
---
Couche 5: dense_11
Configuration: {'name': 'dense_11', 'trainable': True, 'dtype': {'module': 'keras', 'class_name': 'DTypePolicy',
'config': {'name': 'float32'}, 'registered_name': None}, 'units': 12, 'activation': 'softmax', 'use_bias': True,
'kernel_initializer': {'module': 'keras.initializers', 'class_name': 'GlorotUniform', 'config': {'seed': None},
'registered_name': None}, 'bias_initializer': {'module': 'keras.initializers', 'class_name': 'Zeros', 'config':
{}, 'registered_name': None}, 'kernel_regularizer': None, 'bias_regularizer': None, 'kernel_constraint': None,
'bias_constraint': None}
---
```


Figure 35: Configuaration des couches

Avant d'observer les performances du meilleur modèle, on peut commencer par tracer les courbes de perte et de précision de la phase d'entraînement du modèle.

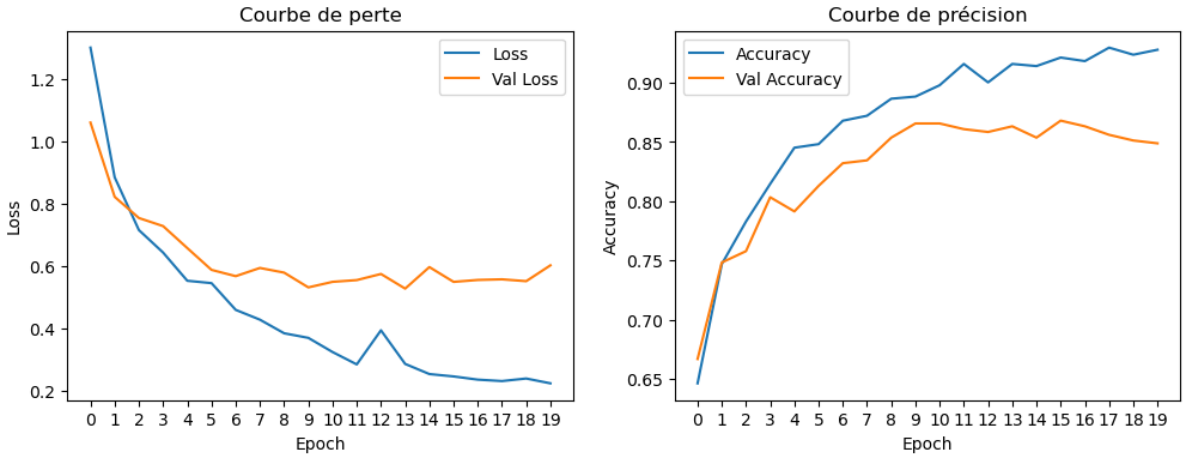


Figure 36: Courbes de perte et de précision

On remarque que les courbes de l'ensemble de validation suivent l'allure des courbes de l'ensemble d'entraînement avant de s'en détacher aux alentours de la 5e époque. La courbe de perte de l'entraînement continue à décroître vers 0 et celle de précision à croître vers 1, alors que les courbes de validation stagnent ou régressent. C'est pour ça que le modèle s'arrêtent à la 20e époque et non la 40e, après plusieurs époques sans diminution de la perte sur l'ensemble de validation. Cet écart entre les courbes indique un effet de surapprantissage des données, le modèle continue de s'améliorer sur les données d'entraînement, mais pas sur celles de validation. A noter que lors de nos essais, les coubes pouvaient être encore plus éloignées et avant la 5e époque. Ce qui indique que ce modèle limite le surapprantissage en comparaison avec les autres modèles.

## 5.3 Résultats du modèle

On affiche la matrice de confusion et les scores obtenus à partir du meilleur modèle de réseau de neurones.

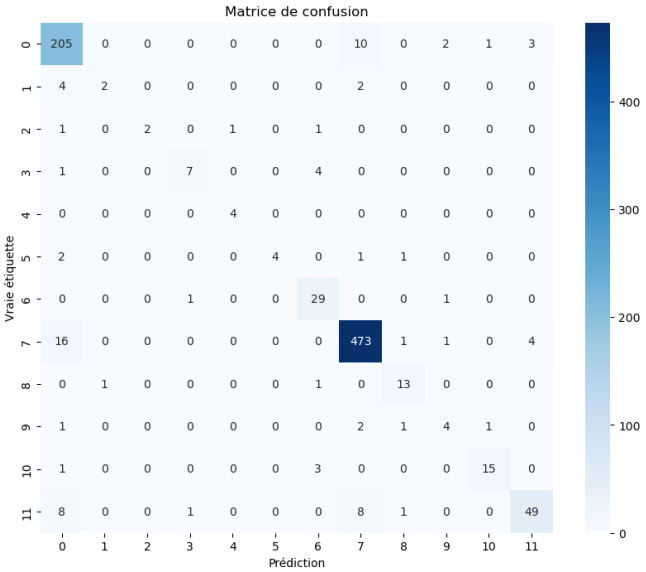


Figure 37: Matrice de confusion réseau de neurones

```
Rapport de classification :
              precision    recall  f1-score   support

           0       0.86      0.93      0.89       221
           1       0.67      0.25      0.36         8
           2       1.00      0.40      0.57         5
           3       0.78      0.58      0.67        12
           4       0.80      1.00      0.89         4
           5       1.00      0.50      0.67         8
           6       0.76      0.94      0.84        31
           7       0.95      0.96      0.95       495
           8       0.76      0.87      0.81        15
           9       0.50      0.44      0.47         9
          10       0.88      0.79      0.83        19
          11       0.88      0.73      0.80        67

    accuracy                           0.90       894
   macro avg       0.82      0.70      0.73       894
weighted avg       0.90      0.90      0.90       894

Précision : 0.9026845637583892
```

Figure 38: Scores de prédiction réseau de neurones

On remarque que notre précision est bien supérieure aux autres méthodes. Elle est aussi plus équilibrée pour chaque classe, avec une meilleure moyenne non pondérée. Le score Recall est aussi meilleur que pour les autres

méthodes, même si certaines classes ont encore une mauvaise performance pour cet indicateur.

# 6 Comparaison

On affiche les scores obtenus par toutes nos méthodes pour pouvoir les comparer, d'abord celles sur le même premier ensemble de test, puis celles sur le test obtenu à partir de l'ensemble des données.

## 6.1 Score sur les méthodes sur 25% des données totales

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Adware | 0.83 | 0.91 | 0.87 | 221 |
| Backdoor | 0.40 | 0.25 | 0.31 | 8 |
| Banker | 0.33 | 0.20 | 0.25 | 5 |
| Dropper | 0.36 | 0.33 | 0.35 | 12 |
| FileInfector | 0.43 | 0.75 | 0.55 | 4 |
| PUA | 1.00 | 0.25 | 0.40 | 8 |
| Ransomware | 0.65 | 0.71 | 0.68 | 31 |
| Riskware | 0.96 | 0.92 | 0.94 | 495 |
| SMS | 0.77 | 0.67 | 0.71 | 15 |
| Scareware | 0.57 | 0.44 | 0.50 | 9 |
| Spyware | 0.54 | 0.74 | 0.62 | 19 |
| Trojan | 0.76 | 0.82 | 0.79 | 67 |
| accuracy | | | 0.86 | 894 |
| macro avg | 0.63 | 0.58 | 0.58 | 894 |
| weighted avg | 0.87 | 0.86 | 0.86 | 894 |

Figure 39: Scores de prédiction k-ppv

Rapport de classification :

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.83 | 0.90 | 0.86 | 221 |
| 1 | 0.29 | 0.25 | 0.27 | 8 |
| 2 | 0.50 | 0.20 | 0.29 | 5 |
| 3 | 0.45 | 0.42 | 0.43 | 12 |
| 4 | 0.50 | 0.75 | 0.60 | 4 |
| 5 | 0.67 | 0.25 | 0.36 | 8 |
| 6 | 0.66 | 0.68 | 0.67 | 31 |
| 7 | 0.96 | 0.92 | 0.94 | 495 |
| 8 | 0.81 | 0.87 | 0.84 | 15 |
| 9 | 0.30 | 0.33 | 0.32 | 9 |
| 10 | 0.70 | 0.74 | 0.72 | 19 |
| 11 | 0.76 | 0.79 | 0.77 | 67 |
| accuracy | | | 0.86 | 894 |
| macro avg | 0.62 | 0.59 | 0.59 | 894 |
| weighted avg | 0.87 | 0.86 | 0.86 | 894 |

Précision : 0.8646532438478747

Figure 40: Scores de prédiction SVM

Rapport de classification :

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.86 | 0.93 | 0.89 | 221 |
| 1 | 0.67 | 0.25 | 0.36 | 8 |
| 2 | 1.00 | 0.40 | 0.57 | 5 |
| 3 | 0.78 | 0.58 | 0.67 | 12 |
| 4 | 0.80 | 1.00 | 0.89 | 4 |
| 5 | 1.00 | 0.50 | 0.67 | 8 |
| 6 | 0.76 | 0.94 | 0.84 | 31 |
| 7 | 0.95 | 0.96 | 0.95 | 495 |
| 8 | 0.76 | 0.87 | 0.81 | 15 |
| 9 | 0.50 | 0.44 | 0.47 | 9 |
| 10 | 0.88 | 0.79 | 0.83 | 19 |
| 11 | 0.88 | 0.73 | 0.80 | 67 |
| accuracy | | | 0.90 | 894 |
| macro avg | 0.82 | 0.70 | 0.73 | 894 |
| weighted avg | 0.90 | 0.90 | 0.90 | 894 |

Précision : 0.9026845637583892

Figure 41: Scores de prédiction réseau de neurones

Il apparaît clairement que le réseaux de neurone est supérieur aux autres méthodes, même en comparaison des arbres et des forêts sur cet ensemble de données que nous n'avons pas mis pour éviter les redites. Son score de précision est meilleur et il évite d'avoir des scores inférieurs à 0.5 pour les petites catégories. L'écart le plus important avec les autres méthodes se fait au niveau de la moyenne non pondérée du recall et du f1-score.

## 6.2 Scores sur toutes les données

On affiche en dessous l'ensemble des résultats trouvés grâce à des méthodes qui ont utilisées 70% de l'ensemble des données en train et 30% des données totales en test.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Adware | 0.90 | 0.89 | 0.90 | 950 |
| Backdoor | 0.41 | 0.44 | 0.42 | 32 |
| Banker | 0.48 | 0.69 | 0.56 | 16 |
| Dropper | 0.49 | 0.56 | 0.52 | 41 |
| FileInfector | 0.43 | 0.46 | 0.44 | 13 |
| PUA | 0.40 | 0.36 | 0.38 | 39 |
| Ransomware | 0.71 | 0.87 | 0.78 | 124 |
| Riskware | 0.95 | 0.95 | 0.95 | 1953 |
| SMS | 0.72 | 0.67 | 0.69 | 64 |
| Scareware | 0.75 | 0.64 | 0.69 | 33 |
| Spyware | 0.92 | 0.72 | 0.81 | 68 |
| Trojan | 0.85 | 0.81 | 0.83 | 275 |
| accuracy | | | 0.89 | 3608 |
| macro avg | 0.67 | 0.67 | 0.67 | 3608 |
| weighted avg | 0.90 | 0.89 | 0.89 | 3608 |

Figure 42: Score de l'arbre

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Adware | 0.90 | 0.93 | 0.92 | 950 |
| Backdoor | 0.72 | 0.56 | 0.63 | 32 |
| Banker | 0.64 | 0.88 | 0.74 | 16 |
| Dropper | 0.92 | 0.59 | 0.72 | 41 |
| FileInfector | 0.86 | 0.46 | 0.60 | 13 |
| PUA | 0.83 | 0.38 | 0.53 | 39 |
| Ransomware | 0.78 | 0.93 | 0.85 | 124 |
| Riskware | 0.95 | 0.97 | 0.96 | 1953 |
| SMS | 0.88 | 0.83 | 0.85 | 64 |
| Scareware | 0.85 | 0.70 | 0.77 | 33 |
| Spyware | 0.90 | 0.82 | 0.86 | 68 |
| Trojan | 0.91 | 0.81 | 0.86 | 275 |
| accuracy | | | 0.92 | 3608 |
| macro avg | 0.85 | 0.74 | 0.77 | 3608 |
| weighted avg | 0.92 | 0.92 | 0.92 | 3608 |

Figure 43: Scores de la forêt

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Adware | 0.88 | 0.89 | 0.88 | 950 |
| Backdoor | 0.26 | 0.25 | 0.25 | 32 |
| Banker | 0.71 | 0.62 | 0.67 | 16 |
| Dropper | 0.51 | 0.54 | 0.52 | 41 |
| FileInfector | 0.47 | 0.54 | 0.50 | 13 |
| PUA | 0.35 | 0.33 | 0.34 | 39 |
| Ransomware | 0.72 | 0.85 | 0.78 | 124 |
| Riskware | 0.95 | 0.94 | 0.95 | 1953 |
| SMS | 0.75 | 0.72 | 0.74 | 64 |
| Scareware | 0.59 | 0.70 | 0.64 | 33 |
| Spyware | 0.88 | 0.78 | 0.83 | 68 |
| Trojan | 0.79 | 0.77 | 0.78 | 275 |
| accuracy | | | 0.88 | 3608 |
| macro avg | 0.66 | 0.66 | 0.66 | 3608 |
| weighted avg | 0.88 | 0.88 | 0.88 | 3608 |

Figure 44: Scores agrégation des deux arbres

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Adware | 0.91 | 0.92 | 0.91 | 950 |
| Backdoor | 0.51 | 0.56 | 0.54 | 32 |
| Banker | 0.64 | 0.88 | 0.74 | 16 |
| Dropper | 0.71 | 0.61 | 0.66 | 41 |
| FileInfector | 0.70 | 0.54 | 0.61 | 13 |
| PUA | 0.50 | 0.36 | 0.42 | 39 |
| Ransomware | 0.74 | 0.92 | 0.82 | 124 |
| Riskware | 0.96 | 0.96 | 0.96 | 1953 |
| SMS | 0.76 | 0.91 | 0.83 | 64 |
| Scareware | 0.64 | 0.76 | 0.69 | 33 |
| Spyware | 0.82 | 0.79 | 0.81 | 68 |
| Trojan | 0.95 | 0.76 | 0.84 | 275 |
| accuracy | | | 0.91 | 3608 |
| macro avg | 0.74 | 0.75 | 0.74 | 3608 |
| weighted avg | 0.92 | 0.91 | 0.91 | 3608 |

Figure 45: Scores agrégation des deux forêts

On voit que la forêt est la méthode avec les meilleurs scores. De plus, le coût en terme de calcul est assez faible, cela nous donne donc un moyen d'avoir un prédicteur efficace à moindre coût.